

A High-Performance and Cost-Effective Hardware Merge Sorter without Feedback Datapath

Makoto Saitoh*, Elsayed A. Elsayed*[†], Thiem Van Chu*[‡], Susumu Mashimo[§], and Kenji Kise*

*Tokyo Institute of Technology, [†]Aswan University, [‡]JSPS Research Fellow, [§]Kyushu University
 {saitoh,abdellah,thiem}@arch.cs.titech.ac.jp, susumu.mashimo@cpc.ait.kyushu-u.ac.jp, kise@c.titech.ac.jp

Abstract—We propose a high-performance and cost-effective hardware merge sorter (HMS) without any feedback datapaths in order to develop the fastest hardware sorting accelerator. The operating frequencies of existing HMSs are severely limited by the presence of feedback datapaths. We show the idea of eliminating the feedback datapaths, and propose a concrete architecture adopting the idea and some implementation optimizations. The evaluation results show that our HMS achieves 1.59x throughput improvement with less hardware resources compared to the state-of-the-art HMS.

Keywords—feedback datapath; FPGA; hardware merge sorter; sorting;

I. INTRODUCTION

Sorting is one of the essential operations used in many areas. With the advent of new technologies such as social networks and IoT devices, there is a significant increase in the size of data to be processed. Therefore, improving the performance of sorting, which has been already researched for several decades, now becomes even more critical.

Besides optimizing sorting algorithms on CPUs [1], [2], many efforts have been made to accelerate the sorting operation through various approaches such as GPU-based accelerators [3], [4]. Recently, there is an increased interest in the FPGA-based sorting architectures [5]–[8]. The state-of-the-art study [8] shows that FPGA-based accelerators can achieve the highest performance compared with optimized algorithms on CPUs with and without GPU-based accelerators. Therefore, using FPGA-based accelerators is becoming a promising approach to improve the performance of sorting compared to others.

Recent FPGA-based sorting accelerators [5]–[8] are based on a basic operation: merging two sorted sequences into one sorted sequence. Like in [8], we call such a sorting accelerator a *hardware merge sorter (HMS)*.

In [5], Koch and Torresen introduced the basic concept of a *merge logic* circuit which merged two sorted sequences stored in FIFOs. This basic merge logic circuit outputs one record per cycle. To achieve higher sorting performance, merge logic circuits that can output multiple records per cycle have been proposed [6]–[8]. We call a merge logic circuit that can output E records each cycle an *E -record merge logic* circuit.

In this paper, the sorted order is assumed to be ascending. A record is composed of two fields: key and satellite data. The order of two records is the order of their keys.

Existing E -record merge logic circuits ($E \geq 2$) can be classified into two models [8]: one was proposed by Casper and Olukotun in [6] and the other was introduced by Song et al. in [7]. They are built based on a sorting network like

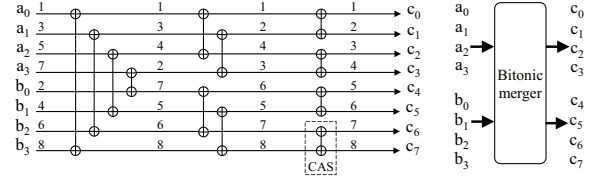


Figure 1: A bitonic merger and its symbol.

a bitonic merger or a Batcher's odd-even merger [9]. This paper adopts the bitonic merger. Thus, we will explain it in detail.

Fig. 1 shows a bitonic merger for merging two sorted 4-record sequences. It is composed of 12 compare-and-swap (CAS) modules. In general, to merge two sorted E -record sequences using a bitonic merger, $E + E \log_2 E$ CAS modules are required. Each CAS module has two input ports and two output ports where the smaller record is output to the upper port while the larger one is output to the lower port. Fig. 1 also shows an example behavior of merging two sorted sequences $\{1, 3, 5, 7\}$ and $\{2, 4, 6, 8\}$. The symbol that we use for a bitonic merger is shown on the right-hand side of the figure.

The throughput of an E -record merge logic circuit is proportional to not only E but also its operating frequency. In [8], Mashimo et al. pointed out that it was hard to make the merge logic model proposed in [7] operate at a high frequency when increasing E because of the complexity of dequeuing and reordering records from the input FIFOs. Thus, we focus on the model proposed by Casper and Olukotun in [6].

To highlight the novel characteristics of our proposed architecture, below, we describe a naive architecture of Casper and Olukotun's model which is our baseline and a much more optimized architecture used by the HMS proposed in [8], namely SHMS.

A. Naive Merge Logic

Fig. 2 shows our baseline E -record merge logic circuit. The components of this circuit are two FIFOs named as FIFO_A and FIFO_B, a comparator, and a core module called *merge network*. Input records are stored in the FIFOs before being fed into the merge network. The smallest record in FIFO_A and the smallest record in FIFO_B are compared, and the FIFO containing the smaller record is selected to dequeue E records.

In the merge network, there are E feedback registers to store the largest E records among all dequeued records. A

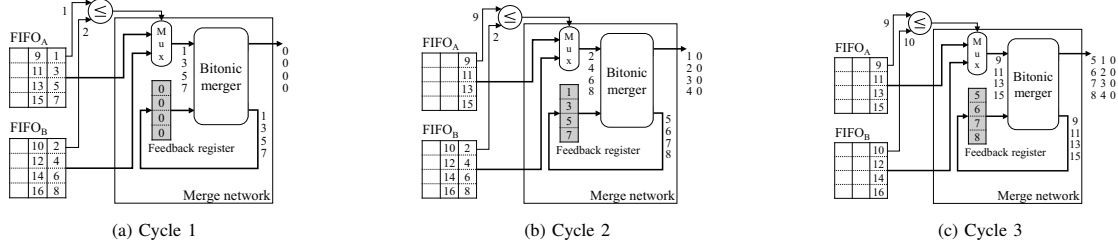


Figure 2: The baseline E -record merge logic architecture for $E=4$ and an example behavior of merging two sorted sequences $\{1, 3, 5, 7, 9, 11, 13, 15\}$ and $\{2, 4, 6, 8, 10, 12, 14, 16\}$.

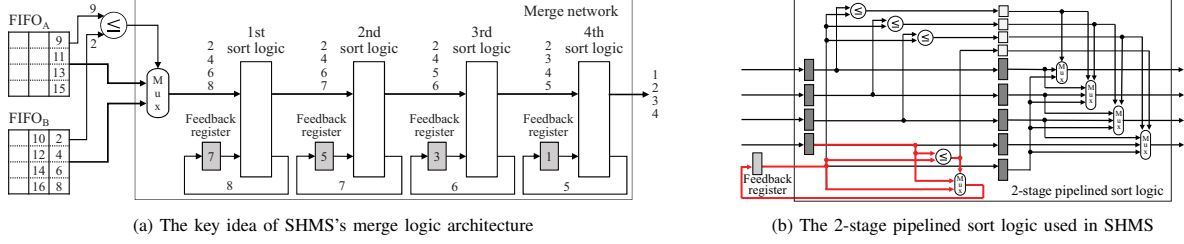


Figure 3: SHMS's E -record merge logic architecture for $E=4$ [8]. The example behavior in (a) is the same as in Fig. 2(b).

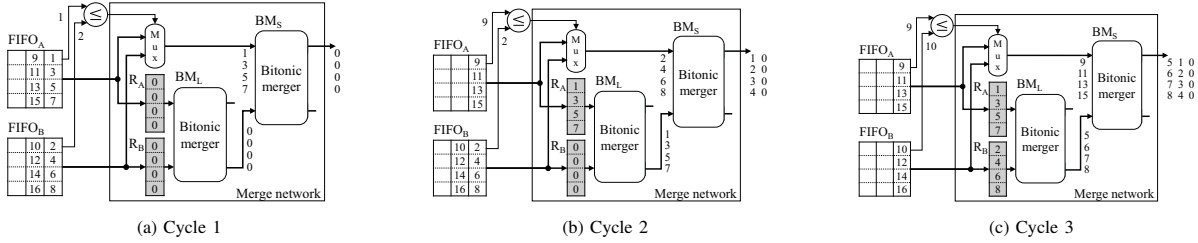


Figure 4: The E -record merge logic architecture for $E=4$ with our **key idea** and an example behavior of merging two sorted sequences $\{1, 3, 5, 7, 9, 11, 13, 15\}$ and $\{2, 4, 6, 8, 10, 12, 14, 16\}$.

bitonic merger merges the two sorted E -record sequences which come from the selected FIFO and the feedback registers.

An example behavior for the $E=4$ merging process in three cycles is shown in Fig. 2. The feedback registers are initialized to the smallest value of all records which is zero in this example.

In Fig. 2(a), the smallest record in FIFO_A and the smallest record in FIFO_B, 1 and 2, are compared, and since $1 \leq 2$ is true, four records $\{1, 3, 5, 7\}$ are dequeued from FIFO_A. These records and four more records $\{0, 0, 0, 0\}$ from the feedback registers are merged by the bitonic merger and the smallest four records $\{0, 0, 0, 0\}$ are output from the merge logic circuit while the largest four records $\{1, 3, 5, 7\}$ are fed back to the feedback registers.

In Fig. 2(b), as the result from cycle 1, the feedback registers hold four records $\{1, 3, 5, 7\}$. Since $9 \leq 2$ is false, four records $\{2, 4, 6, 8\}$ are dequeued from FIFO_B. $\{2, 4, 6, 8\}$ and $\{1, 3, 5, 7\}$ are merged and the smallest four records $\{1, 2, 3, 4\}$ are output and the largest four records $\{5, 6, 7, 8\}$ are fed back to the feedback registers.

In Fig. 2(c), since $9 \leq 10$ is true, the input data of the bitonic merger are $\{9, 11, 13, 15\}$ and $\{5, 6, 7, 8\}$. Thus,

records $\{5, 6, 7, 8\}$ are output and the others are fed back.

In our baseline merge logic circuit in Fig. 2, the *feedback datapath* is the path starting from the feedback registers, passing through the bitonic merger, and returning back to the feedback registers.

The baseline E -record merge logic circuit has a serious drawback: the number of levels of gates increases with increasing E . Its critical path lies in the feedback datapath and consists of $1 + \log_2 E$ sets of a comparator and a 2-input multiplexer. Thus, when E is increased, the operating frequency decreases due to a longer critical path logic delay. To address this drawback, Mashimo et al. [8] proposed a merge logic circuit whose number of levels of gates was constant with respect to E , which will be described in the next subsection.

B. SHMS's Merge Logic

The key idea of SHMS's E -record merge logic circuit ($E=4$) is shown in Fig. 3(a). The merge operation is performed not by a bitonic merger like in the baseline but by a sequence of E sort logic units. Each unit has $E+1$ input records including E sorted records and one record from a feedback register; among $E+1$ input records, the largest one

is fed back to the feedback register while the remaining E are output in sorted order.

The example in Fig. 3(a) is the same as in Fig. 2(b) which illustrates the behavior of the baseline merge logic circuit. The feedback registers hold four records $\{1, 3, 5, 7\}$ and four records $\{2, 4, 6, 8\}$ are dequeued from FIFO_B . We can see that, in both cases, four records $\{1, 2, 3, 4\}$ are output and four records $\{5, 6, 7, 8\}$ are fed back.

The major advantage of the organization in Fig. 3(a) is that the sort logic units can be pipelined in a way that the number of levels of gates of the merge logic circuit is constant with respect to E . Fig. 3(b) shows a 2-stage pipelined sort logic unit introduced in [8]. We can see that using 2- and 3-input multiplexers is sufficient to insert one record into the correct place in a sorted E -record sequence. With this pipelined sort logic, the critical path consists of one comparator and one 2-input multiplexer and is independent of E . Fig. 3(b) highlights the part associated with the critical path. It includes the feedback datapath which starts from and ends at the feedback register.

It is hard to further reduce the critical path. The reason is that the record fed back to a feedback register at any cycle is immediately used at the next cycle and so, we must perform the feedback in one cycle.

The authors of [8] also showed that using just $E - 1$ feedback registers and $E - 1$ sort logic units was necessary and sufficient for correct merge operation. They adopted this optimization in the implementation of SHMS.

C. Our Contributions

While the feedback datapaths are necessary for the merge logic circuits explained in Sections I-A and I-B to operate correctly, they cause two significant problems. First, they are one of the critical paths of these circuits. Second, the presence of feedback datapaths results in complicated wire connections, which may limit the operating frequency.

We propose a novel merge logic circuit without any feedback datapaths. By removing the feedback datapaths, we are able to reduce the critical path of the architecture to one comparator which is simpler than that of the merge logic circuit in [8], and a much higher operating frequency can be expected.

The main contributions of this paper are as follows.

- 1) Most of the existing high-performance merge logic circuits require feedback datapaths for a correct merge operation. Because of this, there is a gap between their throughputs and the ideal throughput that can be achieved. By introducing a way to eliminate the feedback datapaths, we narrow this gap.
- 2) We propose a high-throughput and cost-effective merge logic circuit adopting the idea of eliminating the feedback datapaths. We present a micro-architecture that adopts some implementation optimizations, such as eliminating some control signals that are necessary for previous work.
- 3) We implement an HMS using the proposed merge logic circuit and evaluate it against SHMS, the currently fastest HMS [8], on a Xilinx Virtex-7 FPGA. The evaluation results show that our design achieves 1.59x better peak throughput compared with SHMS

in the significant case the number of records output per cycle is 32 ($E=32$). Our proposal also requires fewer hardware resources when $E=32$ and thus is high-performance and cost-effective.

II. PROPOSED MERGE NETWORK ARCHITECTURE

A. Key Idea for Eliminating Feedback Datapath

The baseline E -record merge network ($E=4$) based on [6] is shown in Fig. 2. Let r_t be the set of E records dequeued from FIFO_A or FIFO_B at cycle t . We define the set of all dequeued records until cycle t as $S_t = \bigcup_{i=1}^t r_i$. Let f_t be the set of E records stored in the feedback registers at cycle t . As explained in Section I, f_t contains the largest E records of S_{t-1} . r_t and f_t are input into the bitonic merger to produce E output records at cycle t and f_{t+1} .

For simplicity, we assume that a record does not contain any satellite data here. Merging records with satellite data requires further considerations which will be discussed in Section II-B.

Our **key idea** for eliminating the feedback datapath is to store $2E$ records dequeued from the two FIFOs in the dedicated registers and provide f_t using only these records in the registers as inputs.

Fig. 4 shows the architecture of the E -record merge network for $E=4$ using the above key idea. Registers R_A and R_B store the most recently dequeued E records from FIFO_A and FIFO_B , respectively. BM_L is a bitonic merger whose inputs are $2E$ records from R_A and R_B . It outputs the largest E records to the other bitonic merger BM_S . The other inputs to BM_S are E records from one of the FIFOs. Among $2E$ input records, BM_S outputs the smallest E records. These E records are the outputs of the merge network. **From the organization shown in Fig. 4, it is clear that the proposed architecture does not have any feedback datapaths.**

To show that the baseline and the proposed architectures are identical regarding their inputs and outputs, firstly, we will show that BM_L provides f_t , the set of the largest E records of S_{t-1} , at cycle t .

Let S_t^A and S_t^B be the sets of records dequeued from FIFO_A and FIFO_B , respectively, from cycle 1 to cycle t . By definition, at cycle t , R_A holds the largest E records of S_{t-1}^A , and R_B holds the largest E records of S_{t-1}^B . Therefore, the largest E records of $S_{t-1}^A \cup S_{t-1}^B$ are in R_A and R_B at cycle t . This is because the local largest E records in R_A are the only candidates of S_{t-1}^A for the global largest E records of $S_{t-1}^A \cup S_{t-1}^B$, and the other local candidates of S_{t-1}^B are only in R_B as well. Using $S_{t-1} = S_{t-1}^A \cup S_{t-1}^B$, we can say that the largest E records of S_{t-1} are in R_A and R_B at cycle t .

Since BM_L outputs the largest E records among $2E$ input records from R_A and R_B , its output data at cycle t are the largest E records of S_{t-1} , that is, f_t .

At cycle t , BM_S takes as its input r_t and f_t , which is the same as the bitonic merger in the baseline merge network in Fig. 2. Therefore, the output data of the two merge networks in Fig. 2 and Fig. 4 are the same. So, we say that the baseline and the proposed architectures are identical regarding their inputs and outputs.

Fig. 4 also shows a behavior example of the proposed architecture using the same input sequences as Fig. 2. We assume that R_A and R_B of the merge network are initialized to the smallest value of all records which is 0 in our example.

In Fig. 4(a), since $1 \leq 2$ is true, four records $\{1, 3, 5, 7\}$ in $FIFO_A$ are sent to R_A and BM_S . BM_L takes eight records $\{0, 0, 0, 0\}$ and $\{0, 0, 0, 0\}$ from R_A and R_B as its input, and thus outputs the largest four records $\{0, 0, 0, 0\}$. Therefore, eight records input into BM_S are $\{1, 3, 5, 7\}$ and $\{0, 0, 0, 0\}$. These records are sorted and the smallest four records $\{0, 0, 0, 0\}$ are output.

In Fig. 4(b), since $9 \leq 2$ is false, four records $\{2, 4, 6, 8\}$ in $FIFO_B$ are sent to R_B and BM_S . R_A contains four records $\{1, 3, 5, 7\}$ sent from $FIFO_A$ at the previous cycle. BM_L outputs the largest four records $\{1, 3, 5, 7\}$ among eight input records. Therefore, the records input into BM_S are $\{2, 4, 6, 8\}$ and $\{1, 3, 5, 7\}$. These are sorted and the smallest four records $\{1, 2, 3, 4\}$ are output.

In Fig. 4(c), the merge operation is performed in the same manner as at the previous cycles. The records in R_A and R_B are $\{1, 3, 5, 7\}$ and $\{2, 4, 6, 8\}$. Thus, BM_L outputs $\{5, 6, 7, 8\}$. BM_S takes as its input eight records $\{9, 11, 13, 15\}$ from $FIFO_A$ and $\{5, 6, 7, 8\}$ from BM_L , and therefore, outputs $\{5, 6, 7, 8\}$.

From the examples shown in Fig. 2 and Fig. 4, we can see that the outputs of the two merge networks are the same with the same input data.

B. Merging Records with Satellite Data

So far, we have assumed that a record does not contain any satellite data. In this subsection, we assume that the records to be merged contain satellite data and some records may have the same key, which will be the general assumption for sorting applications. Firstly, we see the issue, and then we show our solution for it.

Fig. 5 shows an example of a wrong output sequence with a 2-record merge network. Here, each record is denoted by a number followed by an alphabet letter where the number is the key, and the alphabet letter is the satellite data. For instance, record 6A has the key of '6' and satellite data of 'A'. In this example, we use 'x' as don't-care satellite data, and assume that every CAS module in BM_L and BM_S does not swap two records when they have the same key.

We call a record that has the same key with another record a *tie-record*. In the sequences in this example, 6A, 6B, and 6C are tie-records.

At cycle t in Fig. 5(a), BM_L takes four records $\{3x, 4x\}$ and $\{1x, 2x\}$ as its input and outputs to BM_S two records $\{3x, 4x\}$. Note that the records $\{1x, 2x\}$ are not used and discarded. Since the key of record 5x is smaller than that of record 6B, the records $\{5x, 6A\}$ from $FIFO_A$ are selected to be another input to BM_S . Among four input records, BM_S outputs two records $\{3x, 4x\}$. The remaining records $\{5x, 6A\}$ are not used and discarded.

At cycle $t+1$ in Fig. 5(b), among four input records $\{5x, 6A\}$ and $\{1x, 2x\}$, BM_L outputs $\{5x, 6A\}$ and discards $\{1x, 2x\}$. Since $7 \leq 6$ is false, the two records $\{6B, 6C\}$ from $FIFO_B$ are selected to be another input to BM_S . This bitonic merger outputs $\{5x, 6B\}$ and discards $\{6C, 6A\}$. Note that

6A is discarded here since two records 6A and 6B have the same key and the CAS module does not swap them in BM_S .

At cycle $t+2$ in Fig. 5(c), BM_L outputs two records $\{6B, 6C\}$ to BM_S . Since $7 \leq 9$ is true, the two records $\{7x, 8x\}$ from $FIFO_A$ are selected to be another input to BM_S . This bitonic merger outputs $\{6C, 6B\}$ and discards $\{7x, 8x\}$. Note that 6A is discarded again since two records 6A and 6B have the same key and the CAS module does not swap them in BM_L .

We can see that the output sequence $\{3x, 4x, 5x, 6B, 6C, 6B\}$ is wrong since 6A is missing and 6B is output twice. The correct sequence must contain three records of 6A, 6B, and 6C and each record should appear once.

Note that even if we assume that every CAS module in BM_S and BM_L swaps its input records when they have the same key, the issue of missing records still occurs. In this assumption, the merge network in Fig. 5 will output $\{5x, 6A\}$ at cycle $t+1$ and $\{6A, 6C\}$ at cycle $t+2$, and therefore, 6B is missing and 6A is output twice.

We solve this *tie-record issue* by using both the key and the satellite data components when comparing any two records. We assume a 32-bit key field and a 32-bit satellite data field in this paper. Therefore we use 64-bit comparators instead of 32-bit ones. Each 64-bit input of a comparator is composed of the key field on the most significant bit side followed by the satellite data field.

This method enables BM_L and BM_S to distinguish records with the same key but different satellite data. By treating both the original key and satellite data as the new key and comparing the new key, we can virtually regard that the record does not have any satellite data. Therefore, we can solve the tie-record issue.

C. Proposed Merge Network

Fig. 6 shows the proposed *E*-record merge network which is a pipelined implementation of the one shown in Fig. 4. The elimination of the feedback datapath allows us to realize the pipelining easily.

The two-stage **pipelined CAS module** shown in the upper part of Fig. 6 is used in the proposal. In this pipelined CAS module, the comparison is done in the first stage and the swap is done in the second stage. BM_L using the pipelined CAS modules instead of simple ones is called a **pipelined BM_L** . Similarly, BM_S using the pipelined CAS modules is called a **pipelined BM_S** .

Since each pipelined CAS module has two stages, the number of pipeline stages of the pipelined BM_L is $2 \times (1 + \log_2 E)$. For correct timing, the records that go to the pipelined BM_S but do not go through the pipelined BM_L must also traverse $2 \times (1 + \log_2 E)$ stages of pipeline registers. For this purpose, we use the shift register named *SRG*. For instance, in Fig. 6 ($E=2$), BM_L is pipelined in four stages, and thus there are four stages of pipeline registers in SRG for records that do not traverse BM_L . In our implementation, to achieve higher frequency and better resource utilization, we use shift register LUTs (SRLs) to implement the whole or a part of these pipeline registers.

Like BM_L , BM_S is also pipelined in $2 \times (1 + \log_2 E)$ stages. Thus, the total number of pipeline stages of our

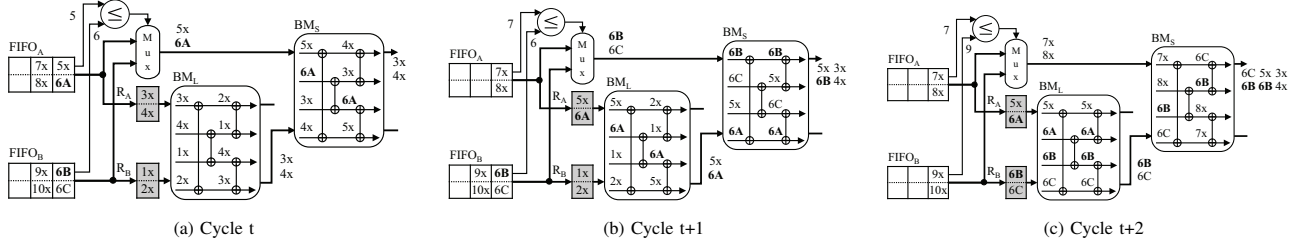


Figure 5: An example behavior of generating a **wrong output sequence** when merging two sorted sequences containing tie-records $\{3x, 4x, 5x, 6A, 7x, 8x\}$ and $\{1x, 2x, 6B, 6C, 9x, 10x\}$. Note that every CAS module does not swap two records when they have the same key.

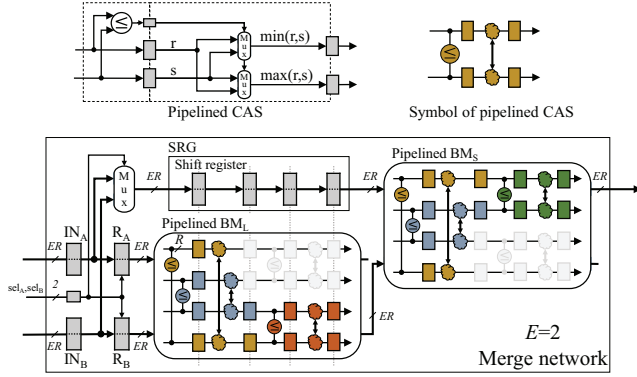


Figure 6: The proposed E -record merge network architecture using SRG and pipelined bitonic mergers.

proposed E -record merge network is $2 + 4 \times (1 + \log_2 E) = 6 + 4 \log_2 E$.

We call a set of E records input/output into/from the E -record merge network in a cycle a *bundle*. The merge network receives a bundle of E records from one of the two FIFOs at its input side ($FIFO_A$ and $FIFO_B$ in Fig. 4). This bundle is latched into a pipeline register IN_A or IN_B .

The signals sel_A and sel_B determine which FIFO the bundle comes from, that is, which FIFO is selected. If sel_A is true, that is, the bundle is from $FIFO_A$, then R_A will be updated. Similarly, if sel_B is true, that is, the bundle is from $FIFO_B$, then R_B will be updated¹.

Our merge network is expected to operate at a high frequency by the following two major reasons. First, it has the shortest critical path length among all existing merge networks. The critical path logic of our merge network is only one comparator² while that of the currently fastest merge network (SHMS's [8]) consists of one comparator and one 2-input multiplexer and is difficult to reduce because of the feedback datapaths with high fan-out feedback registers, our merge network has less complicated wire connections than the others.

Besides the simple critical path logic, our E -record merge

¹We use two select signals because there are cases in which both $FIFO_A$ and $FIFO_B$ are not selected.

²Like in [8], our merge network maintains the feature that the number of levels of gates is constant when E is increased.

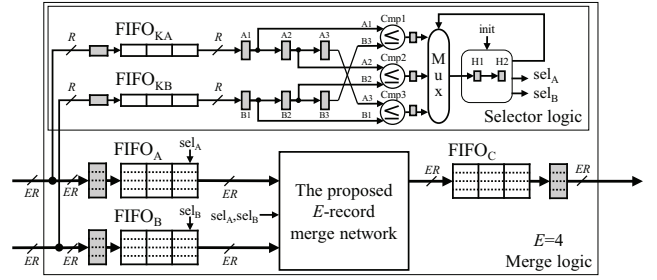


Figure 7: The proposed E -record merge logic architecture.

network has another advantage over SHMS's in terms of logic requirements. Our merge network requires an amount of logic approximately proportional to the number of CAS modules³ which is $2E + E \log_2 E$. In contrast, as described in Section I-B, SHMS's merge network is composed of $E-1$ sort logic units, each has a complexity nearly proportional to E . Thus, it requires an amount of logic approximately proportional to $E(E-1)$. In this way, the logic requirements of our merge network and SHMS's are roughly proportional to $E \log_2 E$ and E^2 , respectively. Therefore, SHMS would require more hardware resources with increasing E .

III. PROPOSED MERGE LOGIC

Fig. 7 shows our proposed merge logic architecture which is constructed using the E -record merge network proposed in the previous section. The role of $FIFO_C$ is described in Section III-B. To select the proper FIFO to be dequeued from $FIFO_A$ and $FIFO_B$, we use a *selector logic* described in the following subsection.

A. Proposed Selector Logic

The architecture of our selector logic is shown in Fig. 7. When a bundle of E records is stored in $FIFO_A$, the smallest record of this bundle is stored in $FIFO_{KA}$. Similarly, the smallest record of a bundle for $FIFO_B$ is stored in $FIFO_{KB}$.

To enable the selector logic to operate as fast as the merge network, we adopt a pipelined design where records dequeued from $FIFO_{KA}$ and $FIFO_{KB}$ are shifted to three

³As mentioned in Section I, a bitonic merger for merging two sorted E -record sequences requires $E + E \log_2 E$ CAS modules. However, each of our bitonic partial mergers (BM_L and BM_S) requires only $E + 1/2 \times E \log_2 E$ CAS modules. Thus, the total number of CAS modules in our E -record merge network is $2E + E \log_2 E$.

stages of pipeline registers (A1, A2, A3; B1, B2, B3)⁴. Let r_{A1}^t denote the record stored in A1 at cycle t . Similarly, $r_{A2}^t, r_{A3}^t, r_{B1}^t, r_{B2}^t$ and r_{B3}^t are defined.

The selector logic computes the **selection result** which should be obtained by the comparison of $r_{A3}^t \leq r_{B3}^t$ at cycle t . Let R^t denote the selection result at cycle t . The signals sel_A and sel_B are generated based on the selection result.

Our architecture of shifting records dequeued from FIFO_{KA} and FIFO_{KB} to three stages of pipeline registers allows us to obtain R^t by using three comparison results two cycles earlier. Then, the obtained two-cycle margin is used for critical path optimizations.

We will explain how R^t can be calculated using earlier comparisons. There are three possible cases depending on the two selection results at cycles $t-2$ and $t-1$. $\text{leftmargin} = *$

- Case 1: If both R^{t-2} and R^{t-1} were true, that is, FIFO_{KA} was selected at both cycles, then A1, A2, and A3 were shifted twice while B1, B2, and B3 were not shifted any time. Thus, we have $r_{A3}^t = r_{A1}^{t-2}$ and $r_{B3}^t = r_{B3}^{t-2}$. Therefore, R^t is obtained from the comparison $r_{A1}^{t-2} \leq r_{B3}^{t-2}$ performed by Cmp1.
- Case 2: If both R^{t-2} and R^{t-1} were false, that is, FIFO_{KB} was selected at both cycles, then B1, B2, and B3 were shifted twice while A1, A2, and A3 were not shifted any time. Thus, we have $r_{A3}^t = r_{A3}^{t-2}$ and $r_{B3}^t = r_{B1}^{t-2}$. Therefore, R^t is obtained from the comparison $r_{A3}^{t-2} \leq r_{B1}^{t-2}$ performed by Cmp3.
- Case 3: If one of R^{t-2} and R^{t-1} was true and the other was false, that is, FIFO_{KA} and FIFO_{KB} were alternately selected at two cycles, then all registers A1, A2, A3, B1, B2, and B3 were shifted once. Thus, we have $r_{A3}^t = r_{A2}^{t-2}$ and $r_{B3}^t = r_{B2}^{t-2}$. Therefore, R^t is obtained from the comparison $r_{A2}^{t-2} \leq r_{B2}^{t-2}$ performed by Cmp2.

By selecting the proper one from the results of Cmp1, Cmp2, and Cmp3 at cycle $t-2$ with a multiplexer, we have the selection result at cycle t .

The history registers H1 and H2 hold the selection results at the most recent two cycles, and the multiplexer is controlled by these registers⁵. As shown in Fig 6, the critical path logic in our selector logic is just one comparator. It thus can operate as fast as the merge network.

B. Stall Signal Elimination for Merge Network (SS_E)

An issue of using most of the existing merge logic circuits in actual practice is that sometimes we have to make them stop outputting records. This is because the logic connected to their output may not always be ready to accept records. SHMS [8] deals with this issue by stalling the update of all registers in the merge network and some other outside registers using a stall signal. While this approach is simple, it makes the FPGA routing task difficult because the stall signal needs to be connected to a vast number of flip-flops (FFs), especially when the merge logic circuit is large.

⁴A similar approach using two stages of pipeline registers are proposed in [8]. Our scheme is an extension of this approach.

⁵The loop structure from the multiplexer in Fig. 7 is feedback. Although our architecture has no feedback datapath, it uses some feedback controls which are not on the critical paths.

In our merge logic circuit, we eliminate the stall signal for the merge network using a FIFO named FIFO_C as shown in Fig. 7. This technique is called *stall signal elimination for merge network* or **SS_E** for short.

Because the merge network is not stalled at all, every bundle that entered the merge network is output some cycles later, and FIFO_C has to keep all of them. To guarantee that FIFO_C does not overflow, the merge logic circuit stops the selector logic dequeuing records from FIFO_A and FIFO_B if necessary.

C. Global Reset Signal Elimination (RS_E)

To sort a large input dataset, we need to perform the merge operation recursively. Thus, after a merge logic circuit finishes merging two sequences, we have to make it ready to accept another two sequences. For example, we have to set all records in R_A and R_B of the merge network to zero as shown in Section II-A.

A naive approach is to set all registers the value zero using a global reset signal. However, this would result in a large number of reset nets, which not only requires a large amount of FPGA routing resources but also may make it difficult to increase the overall operating frequency. The problem becomes more significant with increasing the size of the merge logic circuit.

We propose a mechanism for setting all registers on the datapaths to the value zero without using a global reset signal. In this mechanism, we have a *setup period* before each use of the merge logic circuit. During this period, we input dummy records with the value zero to FIFO_A and FIFO_B and alternately dequeue from them ignoring the comparison results in the selector logic. Then, all registers in IN_A , IN_B , R_A , and R_B in Fig. 6 will be set to zero eventually. When both IN_A and IN_B become zero, the input of SRG becomes zero and thus all registers in SRG will be set to zero eventually. All pipeline registers in BM_L will be also set to zero eventually because its input data are from R_A and R_B which are set to zero by dummy records from FIFO_A and FIFO_B . Similarly, all pipeline registers in BM_S will be set to zero eventually after its input data from SRG and BM_L are set to zero. In this way, dummy records with the value zero gradually sweep the entire datapath of the merge logic circuit and all pipeline registers in R_A , R_B , SRG, BM_L , and BM_S are set to zero after a large enough number of cycles.

We call the above technique *global reset signal elimination* or **RS_E** for short. It is implemented with a minor addition of hardware resources. We verified that RS_E worked correctly using extensive simulations.

IV. EVALUATION AND DISCUSSION

A. Evaluation Setup

Similar to [8], we use the parallel merge tree structure proposed in [7] to evaluate our proposed architectures. In this structure, a merge tree is constructed from merge logic circuits with different sizes increasing from the leaves to the root⁶. We refer to the merge tree with an E -record merge

⁶To adopt the parallel merge tree structure, we modified FIFO_A and FIFO_B in our E -record merge logic circuit to enqueue $E/2$ records per cycle.

Table I: The evaluation summary of the proposed E -record merge tree (MMS) in comparison with SHMS [8]

E	MMS (Proposal)					SHMS [8]				
	Freq. [MHz]	Register	LUT	Active rate	Throughput [Gb/s]	Freq. [MHz]	Register	LUT	Active rate	Throughput [Gb/s]
2	634.9	3,432 (0.28%)	1,629 (0.27%)	0.989	80.4	449.4	1,325 (0.11%)	550 (0.09%)	0.984	56.6
4	606.1	13,900 (1.14%)	6,986 (1.14%)	0.984	152.7	421.1	6,389 (0.52%)	2,664 (0.44%)	0.976	105.2
8	547.9	43,351 (3.54%)	21,831 (3.57%)	0.981	275.2	388.3	24,619 (2.01%)	11,143 (1.82%)	0.972	193.3
16	533.3	121,250 (9.91%)	61,676 (10.08%)	0.977	533.6	344.8	90,127 (7.36%)	44,538 (7.28%)	0.969	342.2
32	506.3	318,942 (26.06%)	164,115 (26.82%)	0.975	1,011.0	320.0	330,684 (27.02%)	175,985 (28.76%)	0.968	634.4

logic circuit at its root as an E -record merge tree.

We call our constructed merge tree *Massive Merge Sorter* (MMS). As a competitor, SHMS [8] was selected because it achieves the highest performance so far. We denote the E -record MMS by MMS(E) and the E -record SHMS by SHMS(E).

For the evaluation, we use four metrics: active rate, FPGA resource utilization, operating frequency, and merging throughput. The active rate is calculated by dividing the number of cycles when valid records are output from the merge tree by the number of elapsed cycles. In this paper, the active rate is defined as α ($0 \leq \alpha \leq 1$) and is evaluated by using a Verilog HDL simulator.

We use the record width of 64-bit where both the key and data fields are 32-bit wide. The record width used in SHMS is 66-bit where the key field is 34-bit including a valid bit and an end-of-stream bit⁷. Since these two bits are only used for control, we do not include them in calculating SHMS's merging throughput. The formula for calculating the merging throughput T (bit/s) of both MMS(E) and SHMS(E) is $T = \alpha \times F \times E \times 64$ where F is the operating frequency in Hz.

For the implementation, we use Vivado 2017.2 instead of Vivado 2016.3 used in [8], and our target FPGA is Xilinx xc7vx980t-2 which is the largest single chip in the Virtex-7 family instead of xc7vx480t-2 used in [8]. We use the largest Virtex-7 FPGA to evaluate the peak performance by minimizing place and route constraints as much as possible. In addition to these changes, we use a global buffer primitive called BUFG explicitly to the reset signal in some SHMS's merge networks. These are the reasons why the results about SHMS in this section are slightly different from those reported in [8]. We use Pblocks to effectively floorplan MMS on the FPGA device to improve the operating frequency.

We use the following fixed parameters. The depths of FIFO_A and FIFO_B are set to 32 in MMS's 2-record and 4-record merge logic circuits and 16 in the other cases. The depth of FIFO_C (only contained in MMS) is set to 32.

We verified that MMS performed the merge operation correctly from large input simulations where each sequence has 1 million records. We confirmed that the output sequence of MMS was identical with the correct sequence obtained from the software version of sorting.

B. Evaluation Results and Discussions

Table I shows the evaluation summary.

1) *Active rate*: To determine the active rate, we simulate the merge operation with randomly generated input sequences, each is 1 million records long as in [8]. In every

⁷MMS does not use these bits for individual records and only uses valid-bit for bundles.

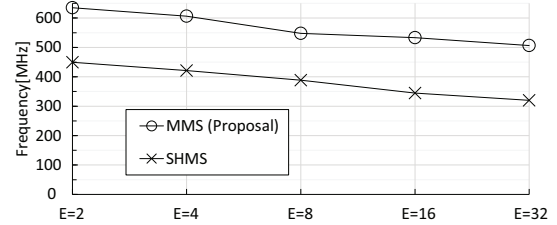


Figure 8: The frequencies of MMS(E) and SHMS(E).

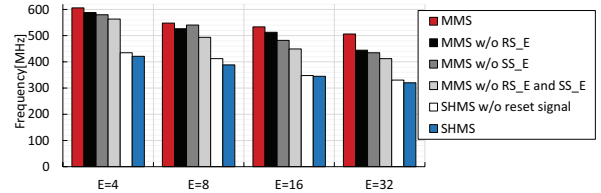


Figure 9: The impact of RS_E and SS_E on MMS and the impact of eliminating reset signal on SHMS.

case, both MMS and SHMS achieve a high active rate of around 0.97 as shown in Table I.

2) *FPGA resource utilization*: As shown in Table I, MMS requires less hardware resources than SHMS for the $E=32$ configuration. Specifically, MMS(32) requires 3.5% less FFs and 6.7% less LUTs than SHMS(32). This result matches with our analysis in Section II-C. The logic requirements of MMS's E -record merge network are roughly proportional to $E \log_2 E$ whereas those of SHMS roughly scale with E^2 . Therefore, MMS is superior to SHMS when E is large.

3) *Operating frequency*: The evaluation results of MMS(E) and SHMS(E) for E from 2 to 32 is depicted in Fig. 8. We can see that MMS is always faster than SHMS thanks to eliminating the feedback datapath and the reset signal, and not stalling the merge network. In particular, MMS(32) meets the constraint of 506 MHz while SHMS(32) only meets the constraint of 320 MHz, which means that the proposal is 1.58x faster than SHMS in their peak frequency comparison.

We next evaluate the effectiveness of our implementation optimizations which are SS_E proposed in Section III-B and RS_E proposed in Section III-C.

The evaluation results of six configurations are summarized in Fig. 9. In this figure, *MMS w/o RS_E* indicates the modified proposal which uses a global reset signal. *MMS w/o SS_E* is the one which uses a stall signal for each merge network component. *MMS w/o RS_E and SS_E* is the one which uses both reset and stall signals. *SHMS w/o reset signal* is the configuration where SHMS's reset signal

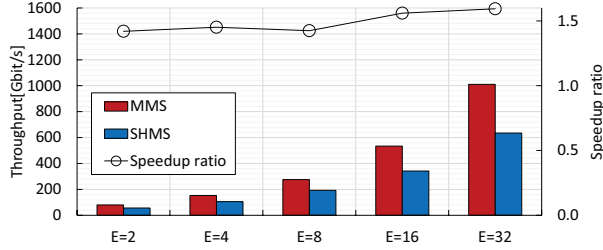


Figure 10: The throughputs of MMS and SHMS with different configurations and the corresponding speedup ratios.

is always de-asserted, which emulates SHMS without a reset signal. Note that evaluating SHMS without stall signals is omitted because eliminating the stall signals for the deeply pipelined architecture in SHMS by using buffers like FIFO_C in Fig. 7 is not reasonable.

For SHMS, the elimination of reset signal only slightly improves its operating frequency, just 2.9% on average. It means that the reset signal elimination technique is still worth to apply for SHMS but does not impact on performance as significantly as the feedback datapath elimination method does.

Now, let us look at the effects of the optimization techniques RS_E and SS_E on MMS. We also can see that not only the feedback datapath elimination but also the two optimization techniques are effective. When $E=32$, although MMS achieves 506 MHz operating frequency, it drops to 444 MHz just without using RS_E optimization. It drops to 434 MHz just without using SS_E optimization. From these comparisons, we say that using these two optimization techniques improves MMS's operating frequency considerably.

4) *Merging throughput*: Because the proposed architecture has a good active rate and a high operating frequency, it can deliver a high throughput as shown in Fig. 10. In this figure, the bar graph indicates the absolute throughput and line graph illustrates the speedup ratio of MMS's over SHMS's.

The best performance configuration where both MMS and SHMS can be fit into the target FPGA device is the case with $E=32$. In this configuration, MMS achieves a throughput of 1,011 Gb/s, while that of SHMS is 634 Gb/s. Therefore, the overall performance of MMS is 1.59x better than SHMS.

From the evaluation results of merging throughput in this sub-subsection and resource utilization in Section IV-B2, we can say that MMS is more cost-effective than SHMS when E is 32.

C. Discussion and Future Work

In [8], it is claimed that not E but $E-1$ feedback registers are necessary and sufficient for an E -record merge network to perform the merge operation correctly. However, we do not use this optimization for the simplicity of implementation. We will use it in future work.

Our solution for the tie-record issue explained in Section II.B is reasonable due to two reasons. First, in applications where each record contains a large amount of satellite data, the satellite data can be replaced by pointers to the records. In this way, the comparator bit-width in our solution is

the sum of the bit-widths of key and pointer, not key and satellite data, and thus large satellite data should not be a serious problem. Second, one of the features of our proposed merge network architecture is that the CAS modules can be pipelined in any number of stages. In this paper, we use the 2-stage pipelined CAS modules considering the tradeoff between the operating frequency and the hardware resource usage. Deeper pipelined CAS modules, where the comparison stage is performed in more than one cycle, may be more suitable for the cases in which the comparator bit-width is larger than 64. The detailed evaluation and discussion of adopting deeper pipelined CAS modules are left as future work.

We believe that the over 1 Tb/s throughput of MMS is highly usable and reasonable for applications on recent hardware like large FPGAs and high-bandwidth memory technologies such as HBM2 whose peak memory bandwidth can be 4 Tb/s [10]. Our next work will be the design and implementation of a sorting accelerator adopting MMS and such high-bandwidth memories.

V. CONCLUSION

We proposed a novel, high-performance and cost-effective hardware merge sorter without any feedback datapaths. Our evaluation reveals that it achieves 1.59x better performance at over 1 Tb/s throughput compared with the state-of-the-art architecture when the number of records output per cycle is 32 while using fewer hardware resources.

ACKNOWLEDGMENT

We sincerely thank James C. Hoe, Tomohiro Yoneda, and Masashi Imai for their helpful comments and feedback.

REFERENCES

- [1] H. Inoue and K. Taura, "Simd- and Cache-Friendly Algorithm for Sorting an Array of Structures," *Proc. VLDB Endow.*, vol. 8, no. 11, pp. 1274–1285, 2015.
- [2] M. Cho, D. Brand, R. Bordawekar, U. Finkler, V. Kurlandaisamy, and R. Puri, "PARADIS: An Efficient Parallel Algorithm for In-place Radix Sort," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1518–1529, 2015.
- [3] D. Merrill and A. Grimshaw, "High Performance and Scalable Radix Sorting: A case study of implementing dynamic parallelism for GPU computing," *PPL*, vol. 21, no. 02, pp. 245–272, 2011.
- [4] A. Davidson, D. Tarjan, M. Garland, and J. D. Owens, "Efficient Parallel Merge Sort for Fixed and Variable Length Keys," in *InPar*, 2012, pp. 1–9.
- [5] D. Koch and J. Torresen, "FPGASort: A High Performance Sorting Architecture Exploiting Run-time Reconfiguration on FPGAs for Large Problem Sorting," in *FPGA*, 2011, pp. 45–54.
- [6] J. Casper and K. Olukotun, "Hardware Acceleration of Database Operations," in *FPGA*, 2014, pp. 151–160.
- [7] W. Song, D. Koch, M. Luján, and J. Garside, "Parallel Hardware Merge Sorter," in *FCCM*, 2016, pp. 95–102.
- [8] S. Mashimo, T. V. Chu, and K. Kise, "High-Performance Hardware Merge Sorter," in *FCCM*, 2017, pp. 1–8.
- [9] K. E. Batcher, "Sorting Networks and Their Applications," in *Spring Joint Computer Conference*, ser. AFIPS, 1968, pp. 307–314.
- [10] D. Manish, S. Jeffrey, and B. Lance, "Intel Stratix 10 MX Devices Solve the Memory Bandwidth Challenge," White Paper, Intel, 2017.